Teaching Hardware Implementation of Neural Networks using High-Level Synthesis in Less Than Four Hours for Engineering Education of Intelligent Embedded Computing

Nan-Sheng Huang
Embodied AI and Neurorobotics Laboratory
Mærsk Mc-Kinney Møller Institute,
University of Southern Denmark, Odense M, Denmark
nan@mmmi.sdu.dk

Jan-Matthias Braun
Embodied AI and Neurorobotics Laboratory
Mærsk Mc-Kinney Møller Institute,
University of Southern Denmark, Odense M, Denmark
j-mb@mmmi.sdu.dk

Jørgen Christian Larsen
Embodied AI and Neurorobotics Laboratory
Mærsk Mc-Kinney Møller Institute,
University of Southern Denmark, Odense M, Denmark
jcla@mmmi.sdu.dk

Poramate Manoonpong
Embodied AI and Neurorobotics Laboratory
Mærsk Mc-Kinney Møller Institute,
University of Southern Denmark, Odense M, Denmark
poma@mmmi.sdu.dk

***Abstract*—This paper presents the motivation and design of a mini-course to teach hardware implementation of neural networks using high-level synthesis (HLS) in less than four hours for engineering education of intelligent embedded computing. By standing on the shoulders of giants, the combination of the real-world problem, decoding the process of neural networks hardware design and using HLS as hands-on lab, the students are able to not only pick up the underlying concepts of digital system design naturally but also implement a real working neural networks hardware accelerator in person. Thus, the main contribution of the work is to facilitate the engineering education of hardware design more engaging and practical.**

***Keywords*—Neural Networks, Hardware Acceleration, FPGA, High-Level Synthesis, Rapid Prototyping, Intelligent Embedded Systems, Engineering Education**

I. Introduction

Machine learning (ML) has within the last 5 years become an important and powerful development within Neural Networks (NNs) for a wide variety of problems in many domains. Those could be pattern recognition, signal processing, image/video processing, classification, prediction, automation and control systems to name but a few [1]. Plenty of NNs algorithms are broadly exploited by software developers and algorithm researchers including Feed-forward Neural Networks (FNNs) or Multi-Layer Perceptrons (MLPs), Radial Basis Networks (RBFs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs) and so on [2]. From the aspect of the embedded application, the capability enables IoT devices to evolve as AIoT (the artificial intelligence of things) which is beneficial to improve the quality and reduce the cost and risk. Thus, it is trending to develop intelligent embedded systems by empowering the devices with the capability of machine learning to process and interpret plenty of noisy and sophisticated measured data sets for further prediction.

Despite its impressive performance, deploying NNs on a low-power real-time embedded system is still a challenging task due to rich computationally expensive operations. To tackle the issue, implementation of hardware-accelerated NNs is a promising solution which can be employed to offload the embedded processor, increase the overall throughput and reduce the latency and energy consumption.

Although Hardware Description Languages (HDLs) such as Verilog, VHDL or SystemVerilog have been extensively used in digital system design for more than 3 decades, it still takes significant efforts and time to implement for the developer with an in-depth understanding of digital hardware design. The designer architects the desired architecture of hardware-accelerated algorithm and then capture the digital circuit behavior by register-transfer level (RTL) or gate-level. RTL assumes to implement the synchronous logic design by abstract logic clouds and registers instead of low-level individual logic gates (NAND, NOR, AND, OR, MUX, FLIP-FLOP) which are less productive. For example, it usually consumes several weeks to develop and verify a handcrafted digital NNs hardware by virtue of RTL as design entry.

From the perspective of engineering education, the steep learning curve not only consumes students much more efforts in developing a hardware-accelerated project but also hinders potential non-hardware background students to learn how to design hardware accelerator to resolve the system performance problems. In order to reduce the entry barrier of hardware design and achieve rapid prototyping, the invention of HLS

978-1-7281-0702-8/19/$31.00

[3] [4] aims to transform the design process of sophisticated digital systems by providing a higher level of abstraction than RTL. HLS leverages software methodologies as hardware design entry and provides many benefits compared to traditional RTL design methodology. For example, HLS takes advantage of the behavioral description in terms of C/C++ to describe the function of hardware-accelerated algorithm together with additional compile directives or pragmas specifying the synthesis constraints in both space and time domain to improve the productivity of hardware design. Hence, it opens the door for more non-hardware background students who are able to write C/C++ programs to design hardware further.

This paper presents the design of a 4-hours mini-course to teach how to design the parallel and pipelined hardware of the offline trained MLPs by the hands-on lab from on-going EU project. The only prerequisite for the students is an elementary knowledge of programming language. In the first hour, the background of HLS-based hardware design is introduced. Next, to draw the attention of students, a real-world embedded brain-computer interface (BCI) research problem using NNs algorithm is taken as an educational case study. Furthermore, MLPs algorithm is introduced to figure out the untimed functional behavior. From the aspect of mathematics, the generic building blocks of MLPs are analyzed. Due to the fact that the MLPs employs a non-linear activation function, a well-known numerical method to approximate the nonlinear function is elaborated. The aforementioned contents are presented in the second hour. In the third hour, a layered architecture is derived by utilizing the analyzed generic building blocks. Next, the implementation is realized by HLS C/C++. Basic HLS C/C++ simulation and synthesis are conducted. Finally, parallel and pipelined methodology supported by HLS is applied to illustrate the process of design-space exploration with quantitative analysis.

Section 2 provides a brief background overview of HLS and high-performance hardware design. The proposed mini-course is described in Section 3. Section 4 discusses the related works. The paper is concluded in Section 5 with future works.

II. Background of HLS-based Hardware Design

In the first hour of the mini-course, the following fundamental concepts of HLS-based hardware design is introduced.

A. Comparison of C/C++ and HLS C/C++

As shown in the left side of Fig. 1, the C/C++ are general-purpose software programming languages executed on top of processors. While programming in C/C++, the code is executed sequentially by the underlying computer architectures. The capability of parallel computations of computer architectures is characterized by various instruction set architectures (ISAs) including single instruction, single data stream (SISD), single instruction, multiple data stream (SIMD), multiple instruction, single data stream (MISD) and multiple instruction, multiple data stream (MIMD). For instance, the ARM NEON instructions are SIMD instructions and the ARM Cortex-M3 processor is SISD architecture. Hence, the SW programmer can only implement the function of algorithm via C/C++ instead of manipulating customized parallelism. However, as illustrated in the right side of Fig. 1, HLS also employs the same C/C++ language but targets to design hardware. The merit of HLS is that it decouples the design for functionality and the design for physical hardware architecture. In HDL-based design, the developer must take into account the two things together before RTL coding. For the SW C/C++ flow, the parallelism is determined by underlying computer architecture instead of customization by the user. In fact, it is a two-pass flow. In pass-1, the user is dedicated to describing the behavior of functionality in HLS C/C++. In pass-2, the user begins to optimize the power, performance, and area (PPA). From user's perspective, it greatly reduces the entry barrier of hardware design by transforming the design of hardware architecture into the use of compiler directives and pragmas.

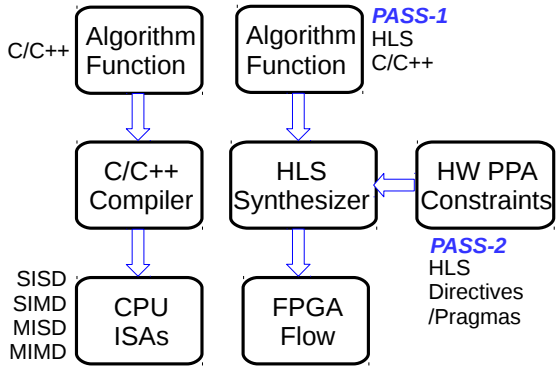

Fig. 1. Comparison of two kinds of C/C++ design flow

B. Parallel and Pipelined Architecture

Parallel and pipelined [5] are design methods of hardware architecture to optimize the PPA. Performance can be represented by throughput and latency. Assume that a task can be allocated and executed by a single processing element (PE) A with a certain execution time B. Parallel involves duplicated hardware resource A in space domain to shorten the execution time B. Pipeline exploits to partition the task into a series of subtasks and therefore subtasks can be paralleled in space domain and chained together in time domain to reduce the execution time B.

C. Computation Bound and Memory bound

Execution of the algorithm mixes both computations such as add, multiply or compare operation and memory access. If the bottleneck results from the insufficient bandwidth of memory access, it is classified as a memory bound issue and vice versa. Array partition is the general method to mitigate the memory bound issue by increasing the ports of memory access in parallel. Fig. 2 demonstrates 3 ways of array partition. Block and cyclic partition directly slice the memory array into multiple pieces where each piece has its

own memory port. The difference is the permutation of the array elements. The behavior of cyclic is interleaved among multiple divided pieces. For complete partition, the whole memory array is divided into individual registers which own the highest parallelism with the price of more area.

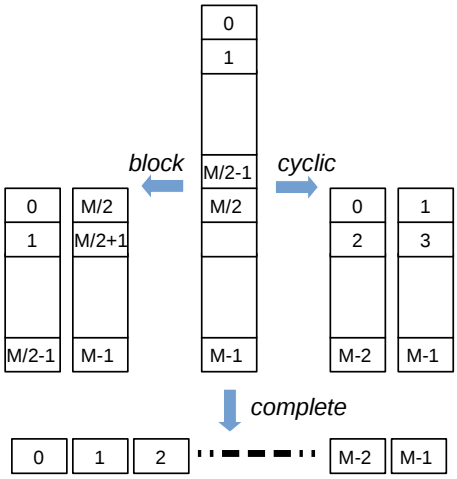

Fig. 2. Example of array partition

D. Design-Space Exploration

Hardware design comprises a variety of methodologies such as pipeline and parallel to achieve target PPA. When it comes to HLS, it facilitates the implementation process to supports each method with a sort of adjustable parameters in either compile directives or pragmas. The design-space exploration is a kind of closed loop system to iteratively sweep possible combination of design parameters to obtain the appropriate result.

III. Proposed Mini-Course

In this section, we elaborate on the strategy of the proposed mini-course to demonstrate how to teach to design a working MLPs hardware in less than 4 hours as shown in Fig. 3.

A. Real World Problem

The on-going Plan4Act project [6] aims to provide new emerging and fascinating embedded technologies that address how neural activity, representing high-level cognitive processes of planning and mental simulation of action sequences can be extracted and used to proactively control the smart home environments. For example, given the use case that a heavily disabled person comes up with a thought to go to the bathroom first while arriving home. In general, a normal person will execute the following three actions in sequences to fulfill the task A-B-C which A means opening the main door, B means switching on the main lights and C means opening the

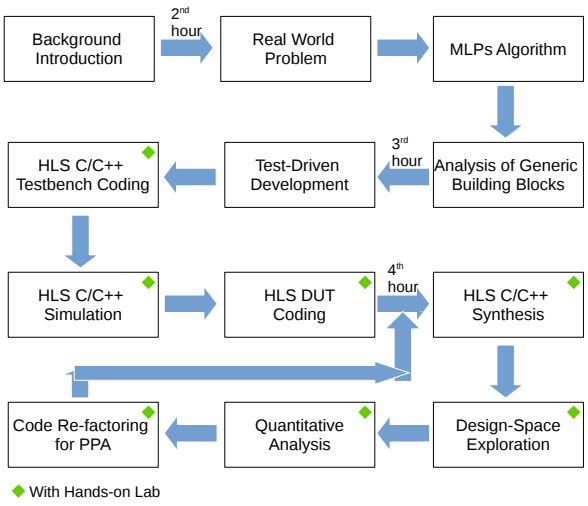

Fig. 3. Proposed teaching trajectory

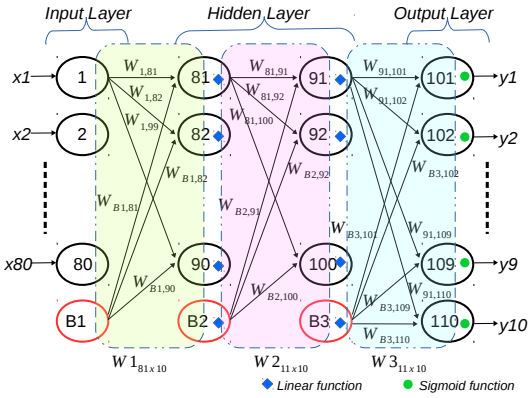

Fig. 4. Architecture of MLPs

bathroom door and light which is a kind of reactive behavior. The Plan4Act project expects to develop a novel proactive control algorithm to predict the complete action sequences A-B-C when obtaining the partial spiking neural data of either A or A-B to predict the corresponding complete action B-C or C proactively. One of the sets of non-linear temporal signal processing algorithm under investigation is MLPs. MLPs are advantageous for hardware implementation due to the simplicity of highly structured network topology, which allows to implement them on FPGA with a minimal area, bandwidth and memory footprint as a real-time embedded device. Therefore, an offline trained MLPs algorithm in the project is leveraged as an educational case study of hardware design.

B. MLPs Algorithm

MLPs is a standard type of feed-forward neural network [7]. As shown in Fig. 4, the selected parameters of offline

trained MLPs comprises 113 neurons distributed to one input layer, two hidden layers, and one output layer. The input layer has 80 pass-through neurons and one bias neuron. The aim of the pass-through neuron is just to transport the external input to the internal layer without any further operation. For the bias neuron, it only emits output with input fixed value one to the next layer. There has a weight connection in between adjacent layers. The MLPs utilizes the topology of full connection. For instance in Fig. 4, all neurons in the input layer is fully connected, meaning that each input neuron connects to all neurons in the next layer. Apart from that, for the neurons located in the hidden layers and output layer, all outputs from all neurons of previous layers are multiplied with associated weight value and then accumulated to dump consolidated output to the neuron. The output of the neuron is then evaluated by the corresponding activation function. In this case, the neurons of hidden layers are equipped with linear transfer function and the neurons of output layers are designed with the sigmoid non-linear function. For bias neurons and input neurons, they do not have an activation function. From the viewpoint of data stream flow, the input data is regularly processed by each neuron and then forwarded to the next layer without any feedback loop. Thus, the computation depends only on the output of the previous layer which implies being beneficial to streamline processing without additional buffers.

$$Y1 = X \cdot W1 = X' \cdot W1' + B1' \tag{1}$$

$$Z1 = f(Y1) \tag{2}$$

$$W1_{80,10} = \begin{pmatrix} w1_{1,81} & w1_{1,82} & \cdots & w1_{1,90} \\ w1_{2,81} & w1_{2,82} & \cdots & w1_{2,90} \\ \vdots & \vdots & \ddots & \vdots \\ w1_{80,81} & w1_{80,82} & \cdots & w1_{80,90} \\ w1_{B1,81} & w1_{B1,82} & \cdots & w1_{B1,90} \end{pmatrix} \tag{3}$$

$$W1'_{81,10} = \begin{pmatrix} w1_{1,81} & w1_{1,82} & \cdots & w1_{1,90} \\ w1_{2,81} & w1_{2,82} & \cdots & w1_{2,90} \\ \vdots & \vdots & \ddots & \vdots \\ w1_{80,81} & w1_{80,82} & \cdots & w1_{80,90} \end{pmatrix}, \tag{4}$$

$$X'_{1,80} = \begin{bmatrix} x1 & x2 & x3 & \cdots & x79 & x80 \end{bmatrix} \tag{5}$$

$$X_{1,81} = \begin{bmatrix} x1 & x2 & x3 & \cdots & x79 & x80 & 1 \end{bmatrix} \tag{6}$$

$$WB1_{1,10} = \begin{bmatrix} w1_{B1,81} & w1_{B1,82} & \cdots & w1_{B1,90} \end{bmatrix} \tag{7}$$

C. Analysis of Generic Building Blocks

From the analysis of Section III-B, it can be inferred that the generic building blocks of MLPs are composed of 3 types of neuron, two types of activation function and plenty of wire connections. However, from the mathematical perspective, it is revealed that the core computational kernel of the MLPs is matrix-vector multiply (MVM) in addition to two types of the activation function. As it can be seen from the Fig. 4, the connections in between layers can be elegantly abstracted by the weight matrix. In virtue of representation of weight matrix, the operation of MLPs can be represented by a series of MVM operation with activation function inserted in between. Furthermore, the building block of MVM is the multiply-accumulate (MAC) unit. The major advantage of MVM compared to the representation by individual neurons is that the more abstract representation is much easier for hardware design thinking and design-space exploration in terms of optimization of PPA. To take an example for illustration, Equation (1) to (7) formulate the operation of the input to hidden layer 1 as shown in the left side of Fig. 4. In (1), the MVM is further split by $X' \cdot W1' + B1'$ to minimize the number of memory access for low power due to the bias neuron only has fixed input value one.

For the building blocks of activation function, the sigmoid is non-linear function and cannot be implemented by hardware directly. Hence, a proper numerical method is a must to be exploited to approximate the non-linear function for further implementation. In this mini-course, 16 segments piece-wise linear (PWL) approximation for the sigmoid function is introduced as shown in Fig. 5. Basic linear line y = ax +b is utilized to approximate the non-linear function and results in the composition of different segments with various parameters a and b. The core building block of PWL approximation consists of comparators, multipliers, and adders which are all primitive logic elements in the hardware library. Fig. 5 demonstrates the associated approximation figure of the sigmoid function. Finally, the hardware architecture of MLPs can be devised as shown in Fig. 6. The ACC-N represents an accumulator unit with the capability of loop bound N in accumulation.

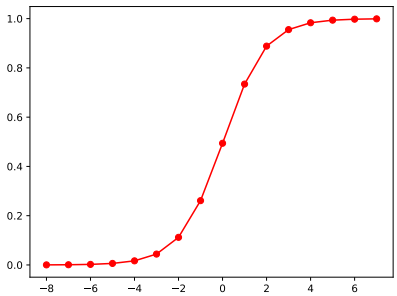

Fig. 5. 16-segments PWL sigmoid function

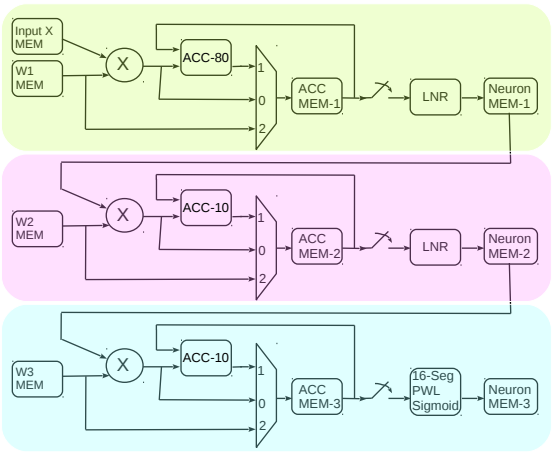

Fig. 6. Layered architecture of MLPs

D. Test-Driven Development (TDD)

TDD is a software development process to pave the road for code refactoring in the later stage which is adopted in the course. Before implementing the MLPs hardware, the student is asked to develop the HLS C/C++ test harness with stub design under test (DUT) and pass some basic test cases first. In this case, provided golden input and output data sets are loaded in the test harness for verification.

Listing 1. HLS Example Code of Single Layer Architecture

```
void mvm_bias(
      data_i_T   data_i[N_IN],
      data_o_T   data_o[N_HID],
      weight_T weights[N_IN+1][N_HID]
)
{
  acc_T    sum=0;

  Outer_MVM:for (int j = 0; j < N_HID; ++j) {
   Inner_MVM:for (int i = 0; i < N_IN; ++i) {
     sum += data_i[i] * weights[i][j];
     if(i==(N_IN-1))
        sum += weights[N_IN][j];
    }// End of Inner_MVM
     data_o[j] = sum;
     sum=0;
  }// End of Outer_MVM

}
```

TABLE I. Example of HLS Directives/Pragmas

Pipeline	#pragma HLS PIPELINE
Parallel	#pragma HLS UNROLL factor=N_PARALLEL
Memory Partition	#pragma HLS ARRAY_PARTITION variable=weights cyclic factor=N_PARALLEL dim=1

E. Design Implementation

According to the above analysis, Listing 1 presents the corresponding HLS C/C++ implementation for the single layer architecture in (1) without activation function. By leveraging the code template, students can quickly build up the complete MLPs in the class. It is quite scalable and modularized which is expected to take less than 15 minutes to implement in HLS C/C++. As can be seen, loops are basic but vital constructs utilized by HLS C/C++ to represent the repetitive algorithmic code. In this case, the repetitive algorithmic code means the fine-grained MACs operation to form the coarse-grained MVM computation as elaborated previously. Furthermore, the implemented DUT can be inserted in the test harness for HLS C/C++ simulation which is significantly faster than RTL simulation.

F. Design-Space Exploration

DSE is achieved by add-on HLS directives/pragmas on top of the developed HLS C/C++ code. Table I shows three associated compile directives in Xilinx Vivado HLS [8] with regard to the hardware design methods introduced in Section II. The value of factor means the degrees of parallelism. For example, first of all, the user needs to specify the name of the variable of HLS C/C++ code to apply the pragmas. Next, if the value of N_PARALLEL is 2, it implies to partition the array into 2 separate memory banks to improve the memory bandwidth in two times for ARRAY_PARTITION directive or to allocate two hardware resources of MACs to support parallel computation in the time domain for UNROLL directive. Finally, the directives are inserted in either right after line 10 or line 9 to explore different design space. In the hands-on lab, students require to explore the DSE via these compiler directives.

G. Quantitative Analysis

Using the 3 compile directives, 9 different designs for the input to the hidden layer of MLPs without activation function are obtained and Table II shows the results of DSE. Next, a series of quantitative analysis is conducted to figure out the rationale behind these figures to seek optimal design parameters. Case 1 is the baseline implementation without any optimization which is an implicit single PE architecture with one DSP. DSP is a kind of dedicated hardware resource for MAC operations in FPGA. It has the latency of 4831 cycles with estimated clocking speed 5.79 ns. In Case 2 and 3, the pipeline is employed in inner and outer loop respectively which results in 5.4X and 11.6X reduction of latency. But it costs with a slight increase of FF/LUT resource in Case 2 and DSP is replaced with much more FF/LUT in Case 3.

Pipeline and unroll loop with factors 2 and 4 are exploited in Case 4 and Case 5. Case 4 has a latency of 481 cycles with estimated clocking speed 7.27 ns which is 2X faster compared to Case 2. However, while the unroll factor is increased to 4, it does not decrease the number of clock cycles as expected by a factor of two. A potential memory bound issue may exist here. According to the evidence revealed from the results of Case 4 and Case 5, the methodology of memory partition is further applied in Case 6 to Case 9. As shown in the Listing 1, there are 3 arrays involved in the computation of MVM

TABLE II. Comparison with Various PPA Optimization Strategies

Case No.	Outer	Inner	Estimated Clock Speed (ns)	Latency (cycles)	BRAM	DSP48E	FF/LUT
1	-	-	5.79	4831	0	1	417/256
2	-	pipeline	5.8	881	0	1	575/329
3	pipeline	-	6.43	416	0	0	8649/8668
4	-	pipeline,unroll=2	7.27	481	0	2	523/469
5	-	pipeline,unroll=4	5.79	481	0	4	764/761
6	-	pipeline,unroll=2,partition=2	7.27	471	0	2	492/405
7	-	pipeline,unroll=4,partition=4	7.27	281	0	4	819/487
8	-	pipeline,unroll=8,partition=8	10.17	191	0	8	1179/630
9	-	pipeline,unroll=10,partition=10	10.17	181	0	10	1292/694

and so that cyclic partition with a factor of N_PARALLEL is applied. Case 6 has a latency of 471 with estimated clocking speed 7.27 ns and consumption of 2 DSPs, 492 FFs, and 405 LUTs. In Case 7, the latency is reduced to 281 cycles with the same estimated clocking speed and consumption of 4 DSPs, 760 FFs, and 761 LUTs. Compared to case 5, the memory bound issue is tackled by a proper factor number of unroll loop together with cyclic array partition. Case 8 has a latency of 191 cycles with estimated clocking speed 10.17 ns and consumption of 8 DSPs, 1179 FFs, and 630 LUTs which is as expected.

Nonetheless, while it increases the factor to 10 in Case 9, the latency does not decrease significantly as expected. This is because the hidden layer has 10 neurons and the factor number 8 almost reaches the theoretical computation bound which 10 neurons perform operations in parallel. Thus, the factor number 10 in Case 9 does not have expected improvement in performance because it is saturated compared to Case 8.

Finally, Table III provides the results of HLS synthesis of the complete MLPs hardware design.

TABLE III. Results of HLS Synthesis of MLPs

Clock Speed	6.92 ns
Latency	1344 cycles
BRAM	0
DSP48E	16
FF	4546
LUT	2957

IV. Related Works and Discussion

Compared to related research in the teaching of hardware design such as [9] [10] [11], most of them are project-based which takes several weeks to approach from bottom-up. Furthermore, the working language of hardware design is HDLs-based which is not as friendly as C/C++ for non-hardware background students. Although in [12] [13] adapted HLS methodology, it still takes much time to build up the real project compared to this work.

The rationale behind the proposed mini-course can be summarized by active learning and active teaching [14]. On one hand, by the introduction of the on-going decoding algorithm design of brain control daily-life tasks in Plan4Act, it reinforces the motivation of students to learn how to design hardware accelerator through involving in real underway project instead of toy case. On the other hand, the use of HLS C/C++ overcomes the high entry barrier of RTL design flow. Students can easily engage with the hands-on labs, materials and participate in the class to implement a working MLPs. It facilitates to learn by in-joy and in-actions.

In the following courses, it is natural to address other design issues such as reusability and scalability. Then, teachers can conduct to explore possible solutions by groups in class. Taking the variation of MLPs as an example, it may evolve into either deep layers which have as many hidden layers as possible or change associated activation function. Moreover, for the number system in underlying hardware, it can be represented by the floating point, fixed-pointed or even logarithmic number system (LNS). C++ template and design pattern method can be introduced to tackle the issue.

V. Conclusion and Future Work

The proposed mini-course makes it possible to utilize on-going BCI research problem and state-of-the-art HLS tool in the design of an MLPs hardware accelerator in less than 4 hours as an opening course of digital system design. Students were given an opportunity to grasp the key concepts of hardware design and further apply to implement an MLPs in class.

In the near future, our plan is to integrate more practical and fascinating problems with HLS hands-on labs into the course such as examples of pattern recognition in Keras. In the meantime, developing the design pattern of the NNs hardware accelerator to build up a class of hardware library of intelligent embedded computing for reuse. Students can discuss directly the design process and elements as well as the user's requirement, helping the hardware design community more readily share.

Acknowledgment

This research is supported by Horizon 2020 Framework Programme (FETPROACT-01-2016FET Proactive: emerging themes and communities) under grant agreement no. 732266 (Plan4Act).

References

[1] T. Kohonen, "An introduction to neural computing," *Neural networks*, vol. 1, no. 1, pp. 3–16, 1988.

[2] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *CoRR*, vol. abs/1705.06963, 2017.

[3] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[4] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[5] K. Parhi, *VLSI Digital Signal Processing Systems: Design And Implementation*. Wiley India Pvt. Limited, 2007.

[6] *EU Plan4Act project*. [Online]. Available: http://plan4act-project.eu/index.php/about/

[7] R. Lippmann, "An introduction to computing with neural nets," *IEEE Assp magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[8] Xilinx, *Vivado Design Suite User Guide: High-Level Synthesis*, Feb,2017. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug902-vivado-high-level-synthesis.pdf

[9] T. Sansaloni, A. Pérez-Pascual, V. Torres, V. Almenar, J. F. Toledo, and J. Valls, "Fft spectrum analyzer project for teaching digital signal processing with fpga devices," *IEEE Transactions on education*, vol. 50, no. 3, pp. 229–235, 2007.

[10] A. Rodríguez, J. Portilla, E. de la Torre, and T. Riesgo, "Teaching hybrid hw/sw embedded system design using fpga-based devices," in *Design of Circuits and Integrated Systems (DCIS), 2016 Conference on*. IEEE, 2016, pp. 1–5.

[11] V. Bonato, M. M. Fernandes, J. M. Cardoso, and E. Marques, "Practical education fostered by research projects in an embedded systems course," *International Journal of Reconfigurable Computing*, vol. 2014, p. 7, 2014.

[12] I. Skliarova, V. Sklyarov, A. Sudnitson, and M. Kruus, "Integration of high-level synthesis to the courses on reconfigurable digital systems," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*. IEEE, 2015, pp. 166–171.

[13] D. Navarro, O. Lucia, L. A. Barragan, I. Urriza, and J. I. Artigas, "Teaching digital electronics courses using high-level synthesis tools," in *e-Learning in Industrial Electronics (ICELIE), 2013 7th IEEE International Conference on*. IEEE, 2013, pp. 43–47.

[14] J. Grunert, *The course syllabus: A learning-centered approach*. Boston, MA: Anker Publishing Co, Inc, 1997.